Machine Learning: Selected Topics

ITI0210, lecture 15 (2021)

Review

Neural networks:

gradient, backpropagation, power of layers

Reinforcement learning:

learn from reward and punishment

Machine Learning Problems

Problem	Explanation	Examples
supervised learning	training examples with the “correct answer” are available or can be created	image recognition with convolutional neural networks
reinforcement learning	no labeled examples, can detect if something goes right/wrong	robot crawling with Q-learning
unsupervised learning	no direct way of telling what the correct result is	mapping behavior of tourists in a city

Unsupervised learning overlaps with data mining,
will not cover in this course

Today’s Topics

- k-nearest neighbors
- decision trees
- linear regression
- recurrent neural networks
- deep reinforcement learning

Nearest Neighbors

Learning without a Model

k-Nearest Neighbors

Simple, flexible, powerful

To classify a new data point p

- Find k closest data points
- Class of p is the *mode* (most common) of these

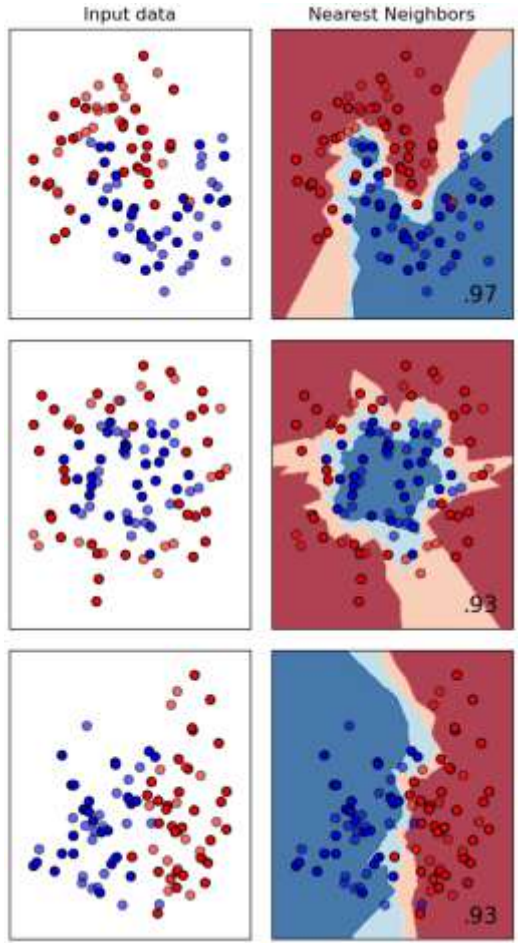
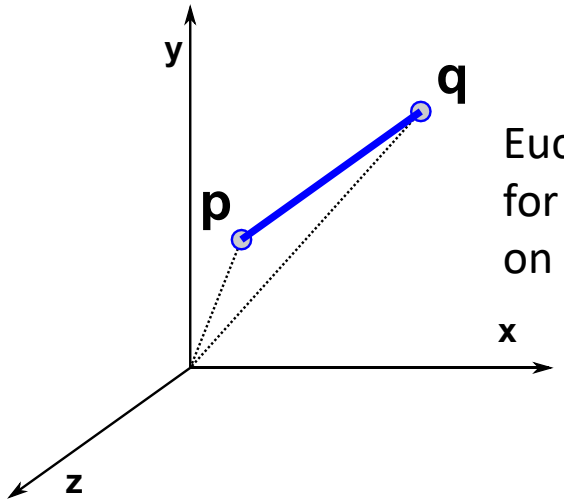


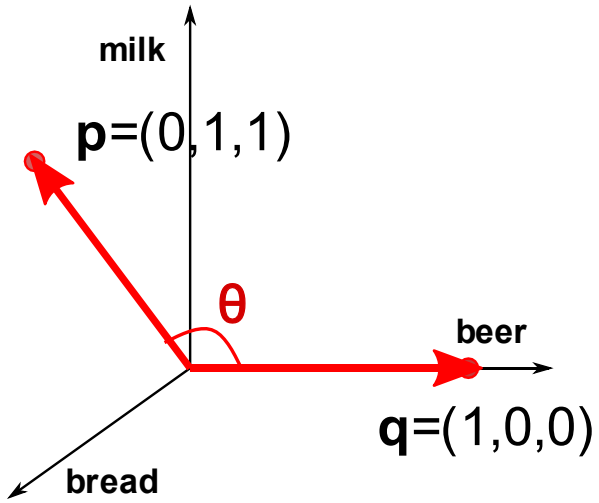
Image: scikit-learn.org

Distance Measures

"Nearest" implies measuring distance



Euclidean:
for numeric features
on comparable scale

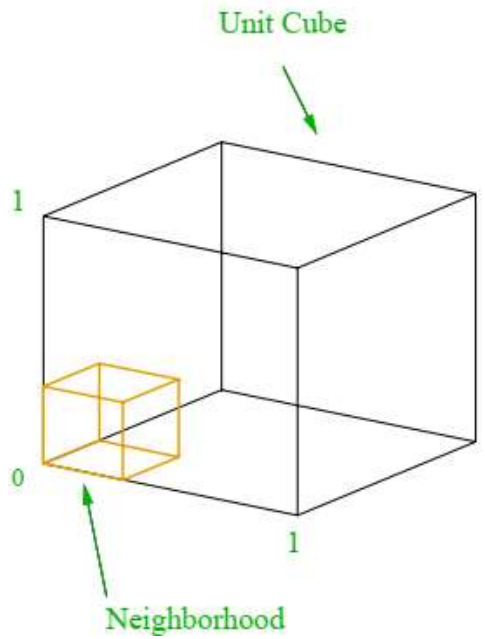


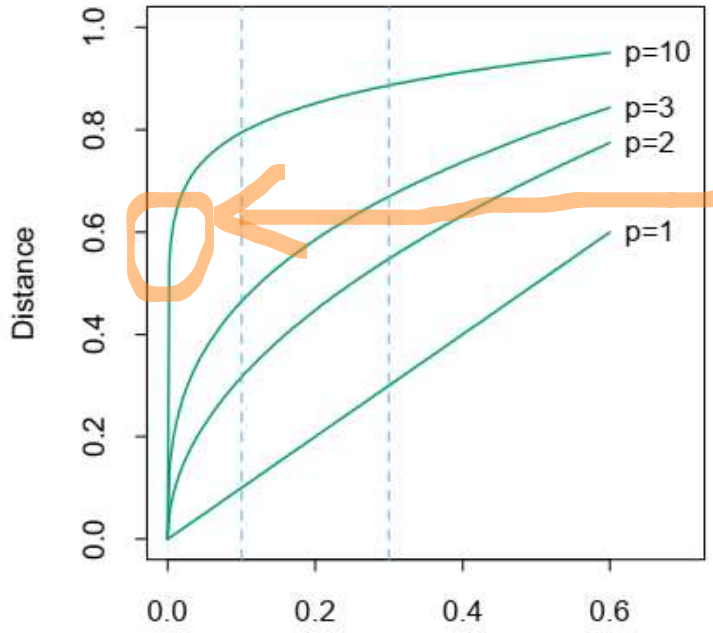
Cosine: $1 - cos\theta$
Texts, sets etc.
Shown is distance
between shopping
baskets p and q

May need a spatial index (K-D tree, Ball Tree) to find neighbors fast

Curse of Dimensionality

Need to include more dissimilar data points to get k-neighborhood in high dimensions





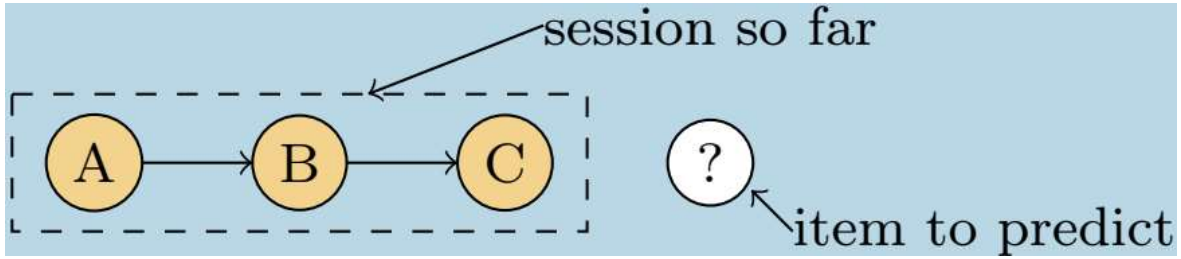
Even closest points may be quite dissimilar (up to 0.6) in **every** feature

Assuming 0 – similar and 1 – dissimilar for each feature

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer 2009.

Performance

k-NN can have high performance:



Equal performance to deep learning measured when predicting

e.g item views and purchases in 4 different e-commerce datasets

Ludewig, Malte, and Dietmar Jannach. "Evaluation of session-based recommendation algorithms." *UMUAI* 28.4-5 (2018): 331-390.

Decision Trees

Information Content of Features

The Restaurant Example

Alt: alternative nearby

Res: have reservation

Est: estimated waiting time

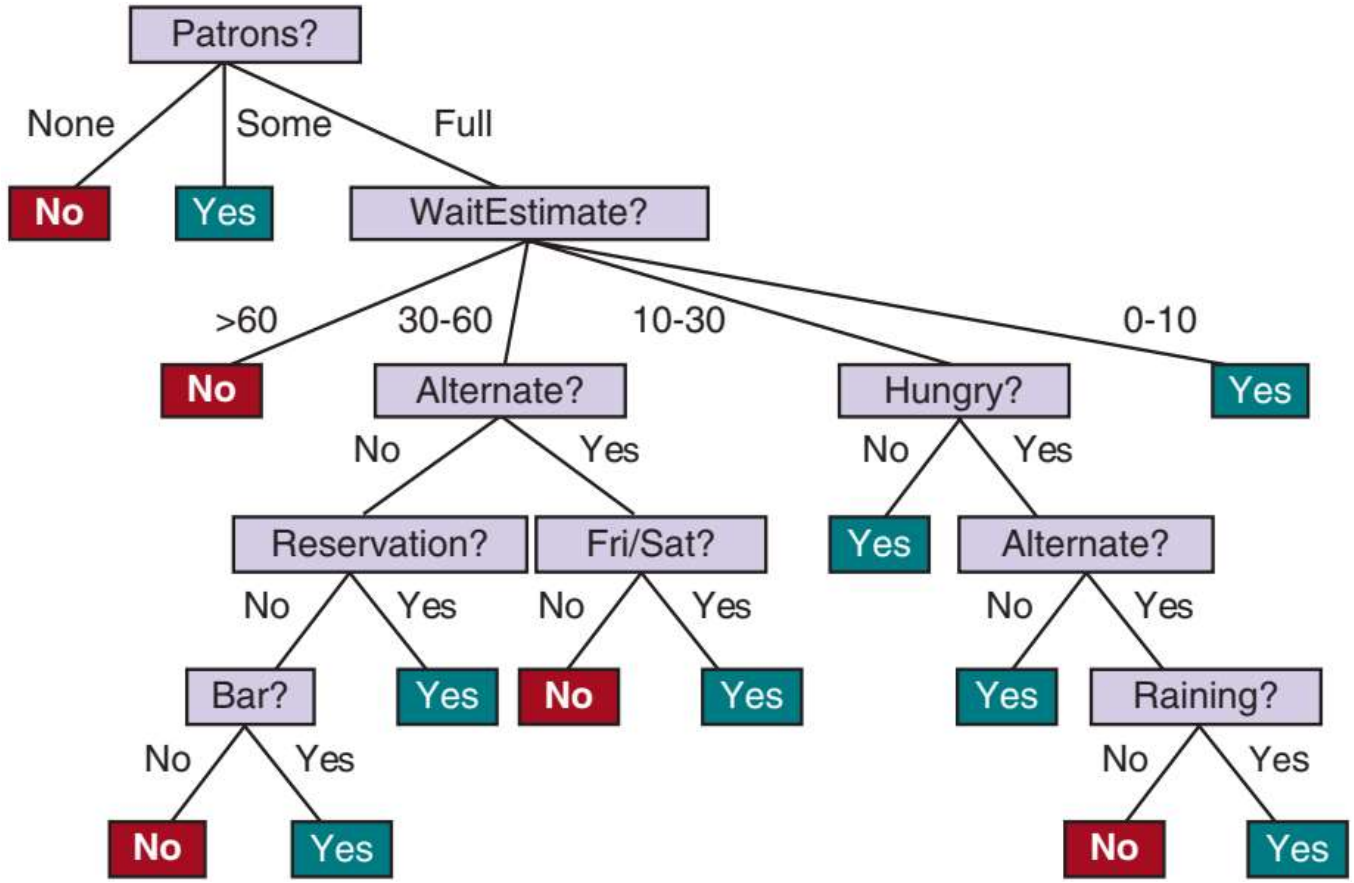
Will Wait:
the attribute
to predict

Alt	Bar	Friday	Hungry	Patrons	Price	Rain	Res	Type	Est	Will Wait
Yes	No	No	Yes	Some	$$$	No	Yes	French	0-10	Yes
Yes	No	No	Yes	Full	$	No	No	Thai	30-60	No
No	Yes	No	No	Some	$	No	No	Burger	0-10	Yes
Yes	No	Yes	Yes	Full	$	No	No	Thai	10-30	Yes
Yes	No	Yes	No	Full	$$$	No	Yes	French	>60	No
No	Yes	No	Yes	Some	$$	Yes	Yes	Italian	0-10	Yes
No	Yes	No	No	None	$	Yes	No	Burger	0-10	No
No	No	No	Yes	Some	$$	Yes	Yes	Thai	0-10	Yes
No	Yes	Yes	No	Full	$	Yes	No	Burger	>60	No
Yes	Yes	Yes	Yes	Full	$$$	No	Yes	Italian	10-30	No
No	No	No	No	None	$	No	No	Thai	0-10	No
Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30-60	Yes

A human decision tree

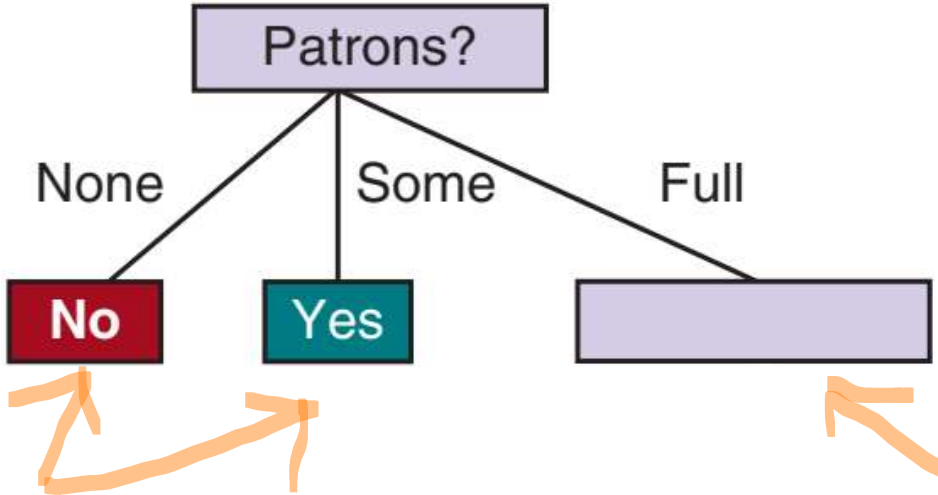
Makes correct decisions for the dataset

We can learn a tree like this **automatically**



Building a Decision Tree

Start somewhere: pick an attribute, look at possible values

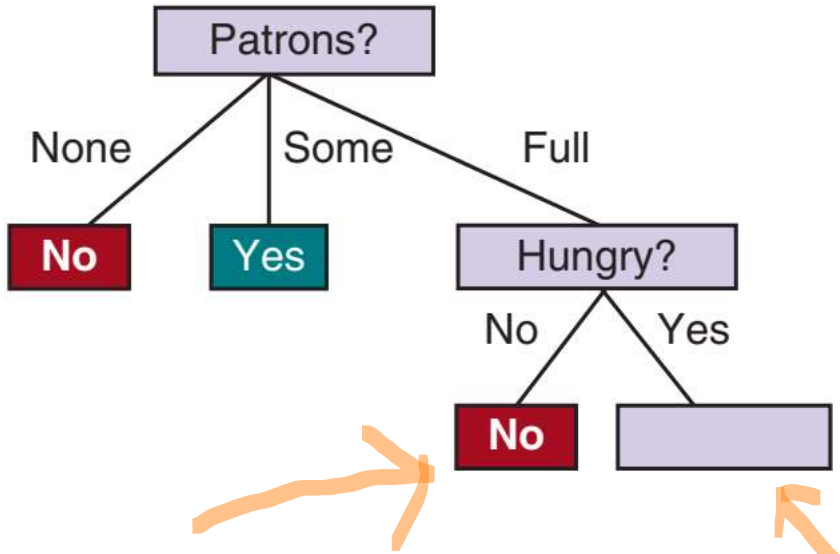


Can decide immediately:
all cases with “None” and “Some”
have the same “WillWait” value

Decision not clear yet –
need to check more attributes

Building a Decision Tree

Next attribute:



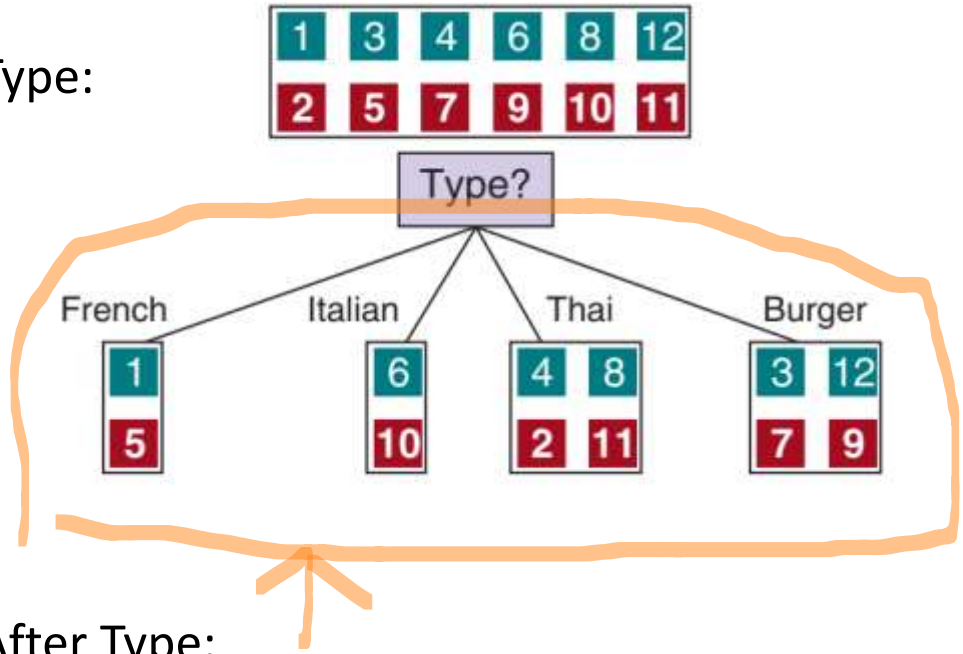
Patrons = “Full” and Hungry = “No”:
decision possible (WillWait = “No”)

Need more attributes
to decide

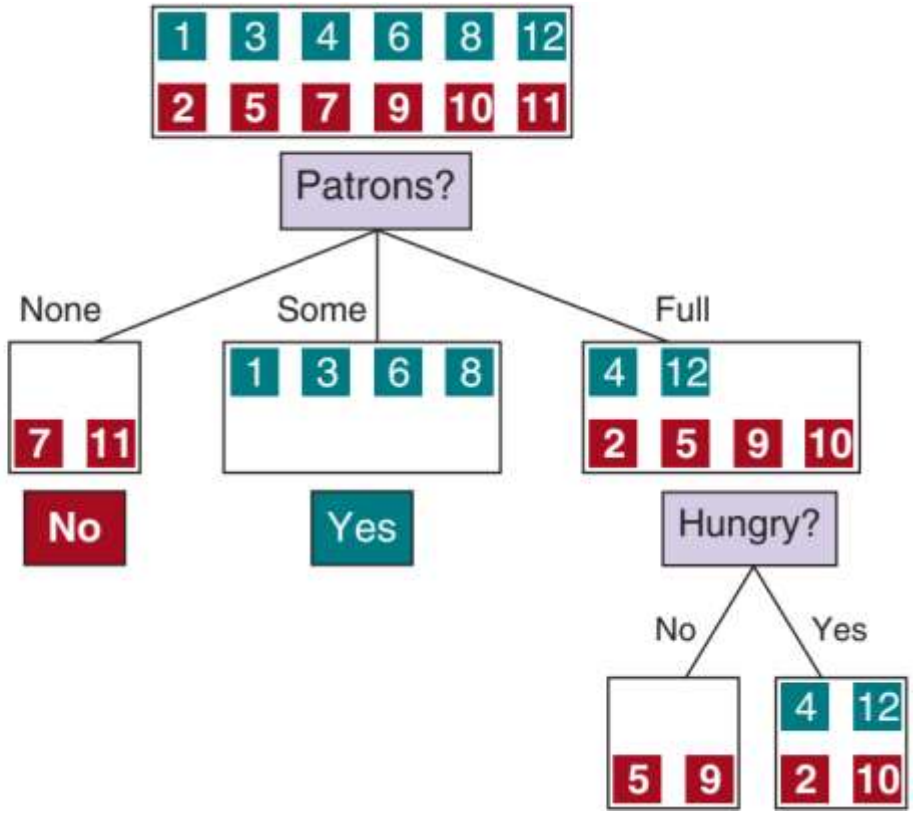
Information Content

Choosing the first and next attributes: compare the predictive power

Before Type:
50/50



After Type:
each branch still 50/50,
no new information



Information Gain

Entropy of information: how many bits of unknown information

For random variable X, $\quad H(X) = -\sum_x P(x) \log_2 P(x)$

Amount of new information learned:

"result of coin toss was tails"

$$H(Tails) = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 1$$

more predictable, less information

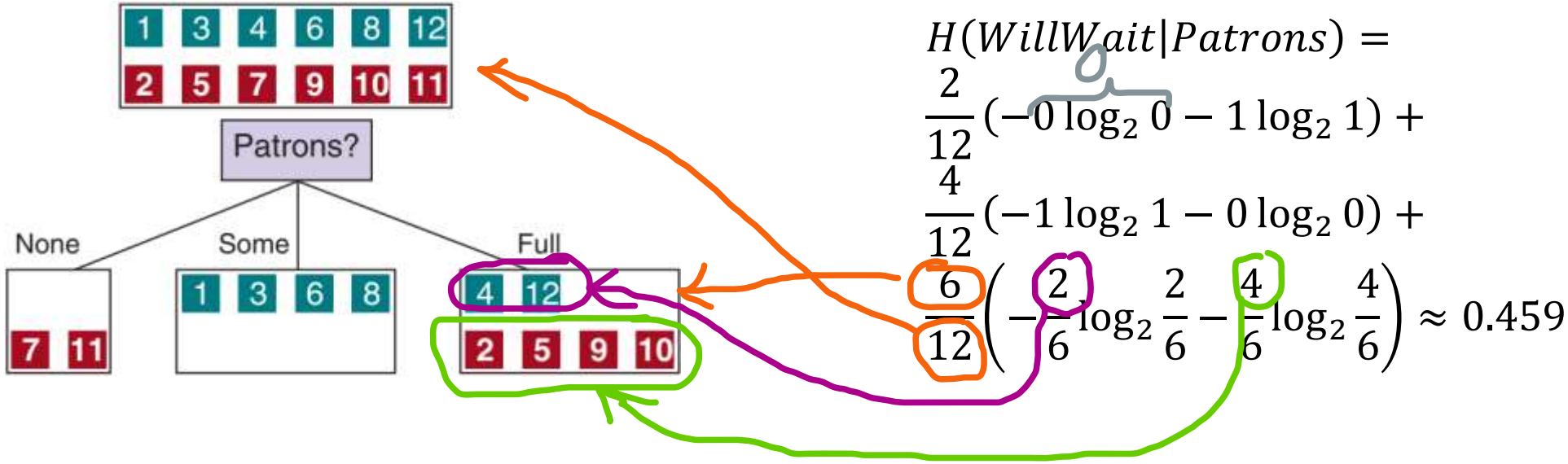
"two people taking the AI course have the same birthday"

$$H(SB) = -(0.999 \log_2 0.999 + 0.001 \log_2 0.001) \approx 0.01$$

Information Gain

Conditional entropy (when Y is known)

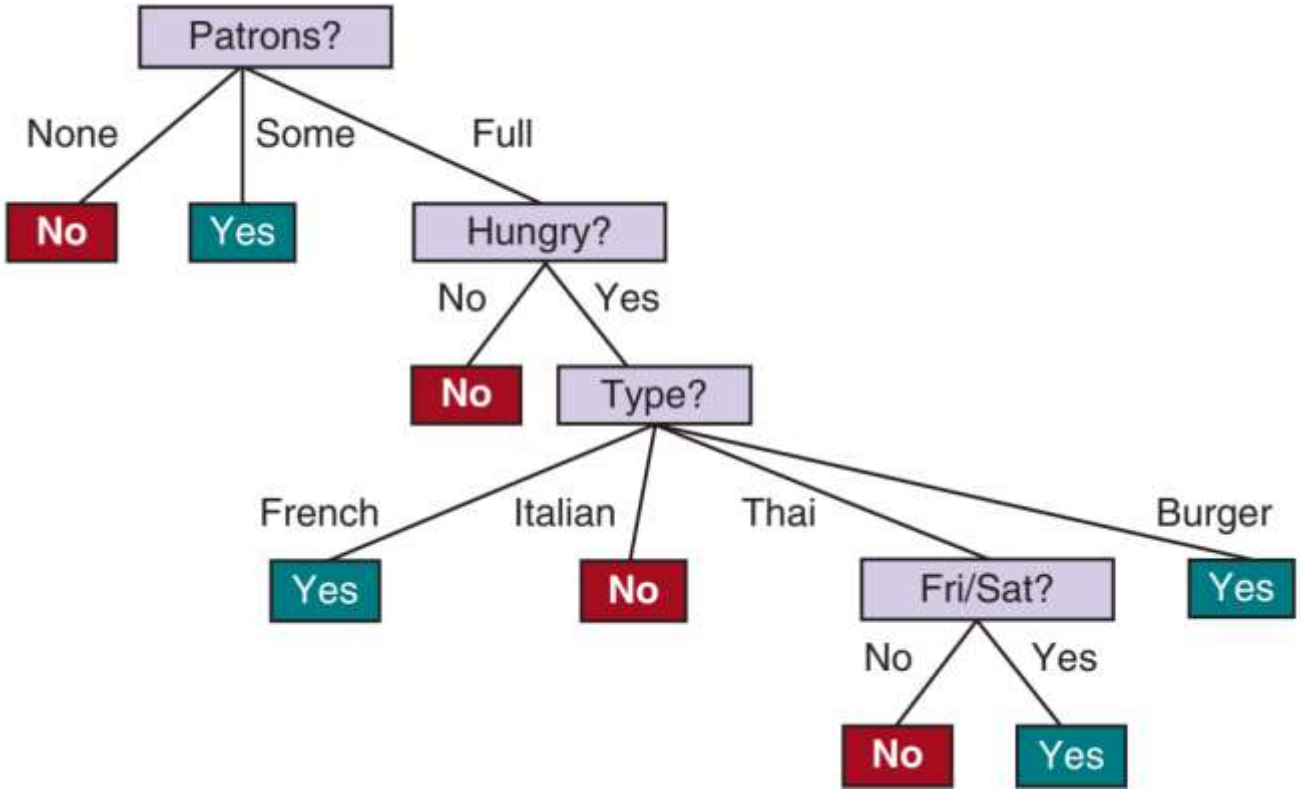
$$H(X|Y) = \sum_y P(y) \left(-\sum_x P(x|y) \log_2 P(x|y) \right)$$



$$H(WillWait|Patrons) =$$
$$\frac{2}{12}(-0 \log_2 0 - 1 \log_2 1) +$$
$$\frac{4}{12}(-1 \log_2 1 - 0 \log_2 0) +$$
$$\frac{6}{12}\left(-\frac{2}{6}\log_2 \frac{2}{6} - \frac{4}{6}\log_2 \frac{4}{6}\right) \approx 0.459$$

The Restaurant Example

Automatically learned tree:

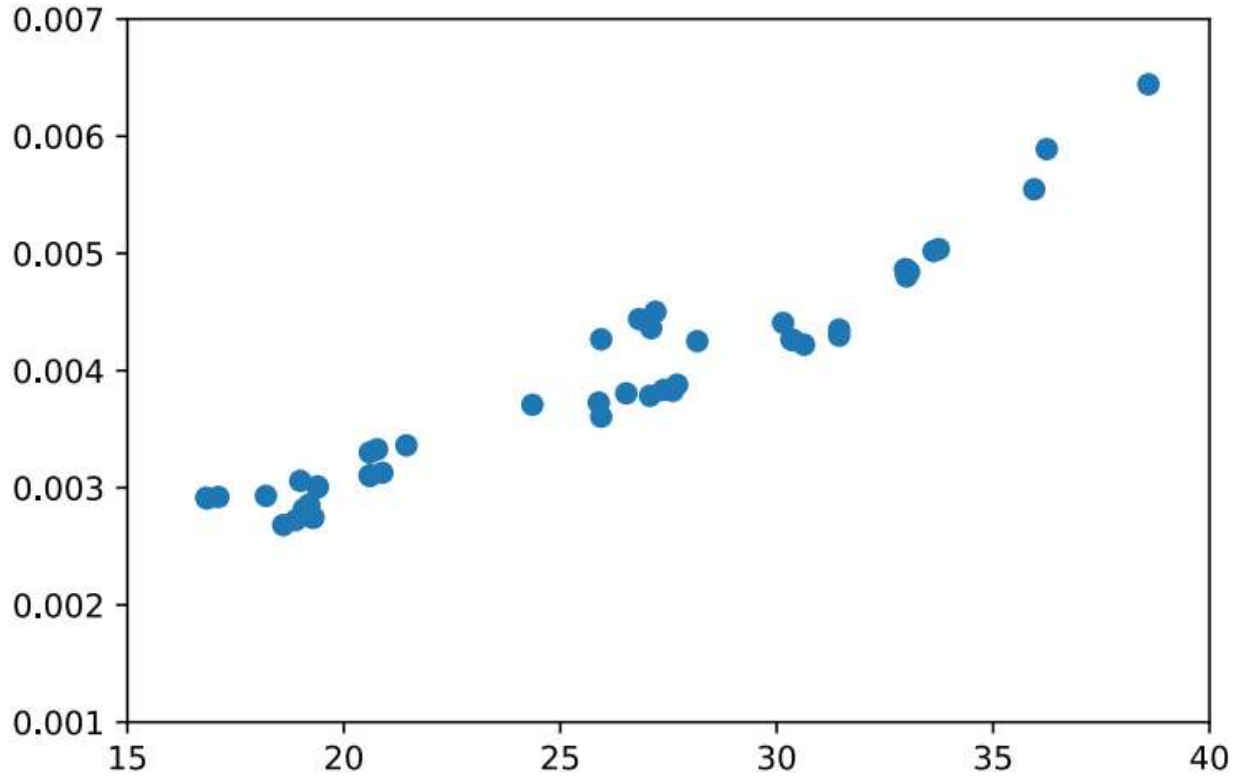


Linear Regression

Fitting Data to Simple Models

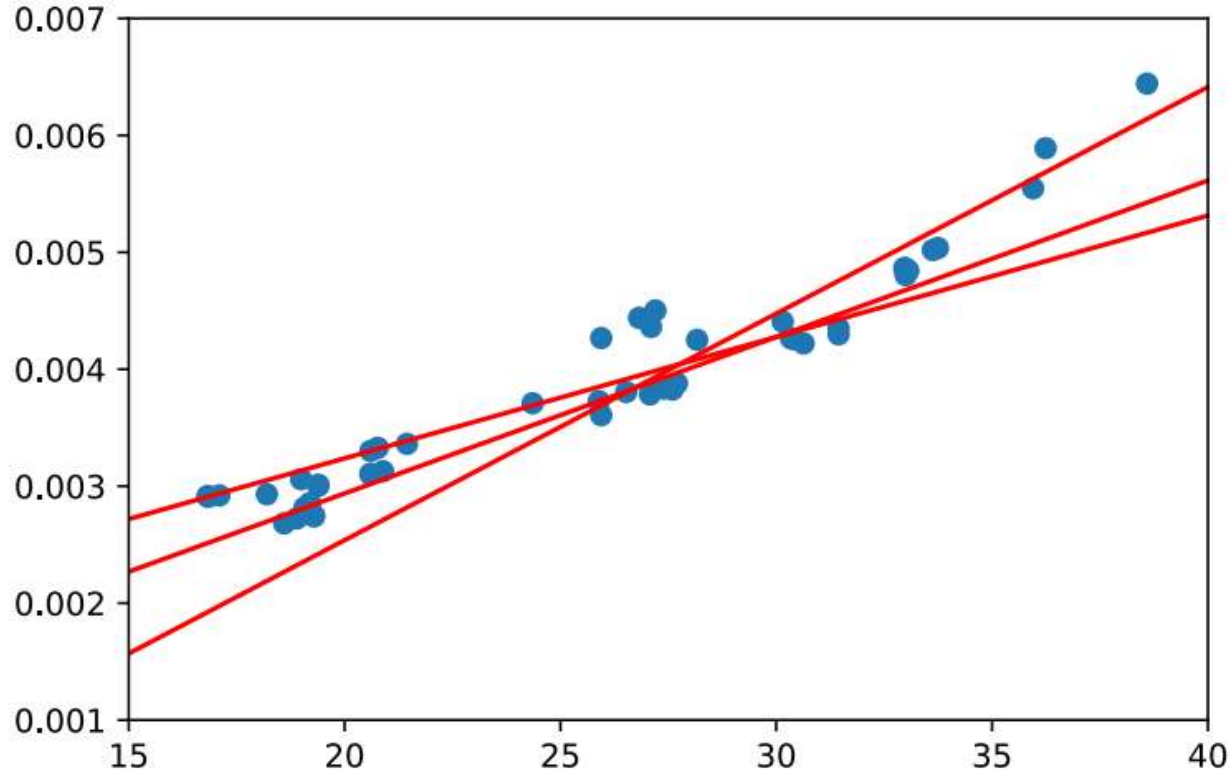
Correlated Data

Looks like we can predict y from x here:



Linear Model Fit

Linear relation is a good guess, but which of these lines is correct?



Linear Model Fit

Our hypothesis:

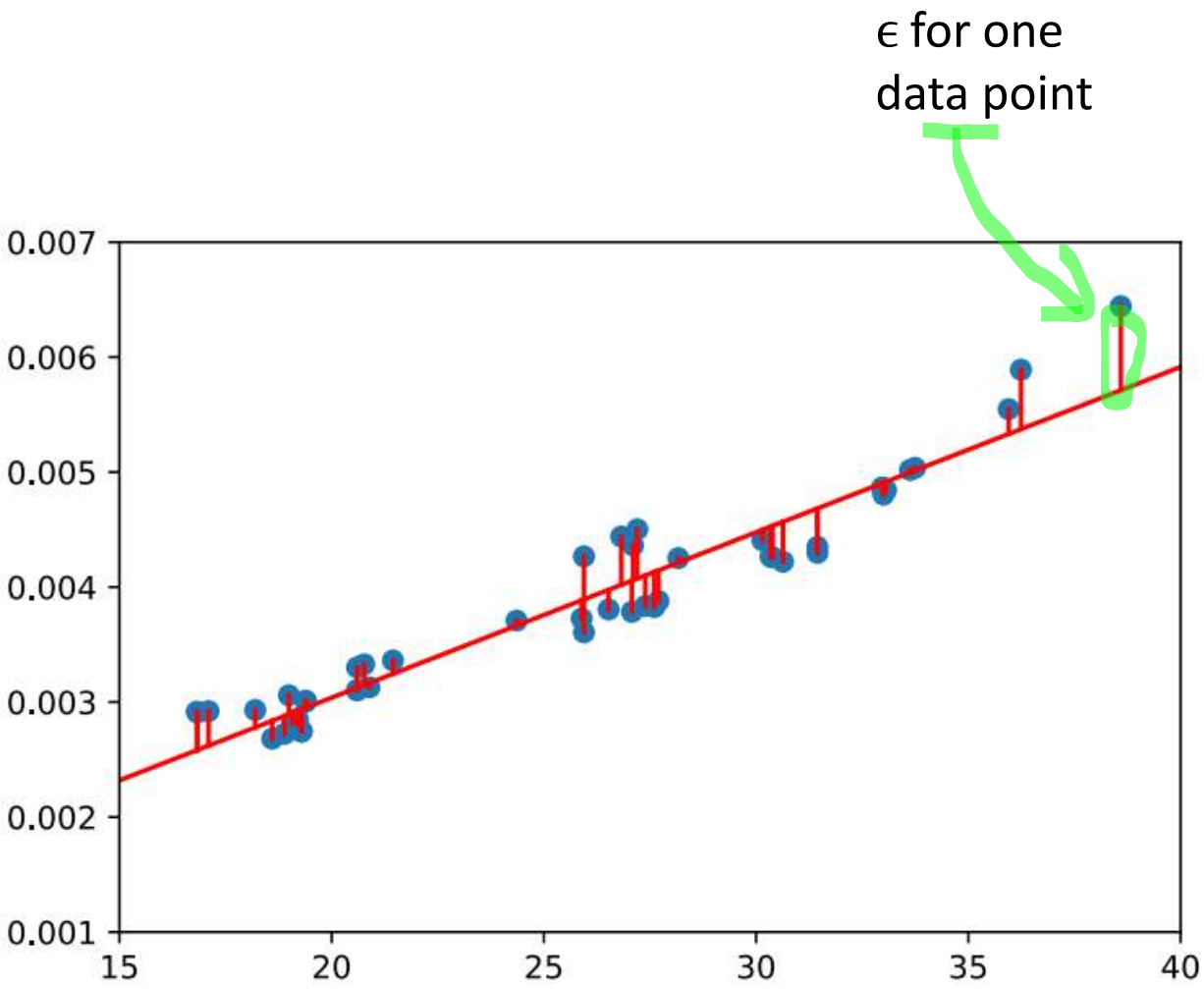
$\hat{y} = w_0 + w_1 x$

True function:

$y = w_0 + w_1 x + \epsilon$

Approach:

Choose w_0, w_1 to **minimize error** ϵ



Finding Parameters

Loss function represents how good our approximation is overall

$$L = \sum_{j}^{N} (y_j - \hat{y}_j)^2$$

Calculate over N training samples

Method 1: gradient descent (repeat until loss stops decreasing):

$$w_i = w_i + \alpha \sum_{j}^{M} (y_j - \hat{y}_j) x_{i,j}$$

$$w_0 = w_0 + \alpha \sum_{j}^{M} (y_j - \hat{y}_j)$$

Can use a batch of M samples, $M = N$ is classical gradient descent

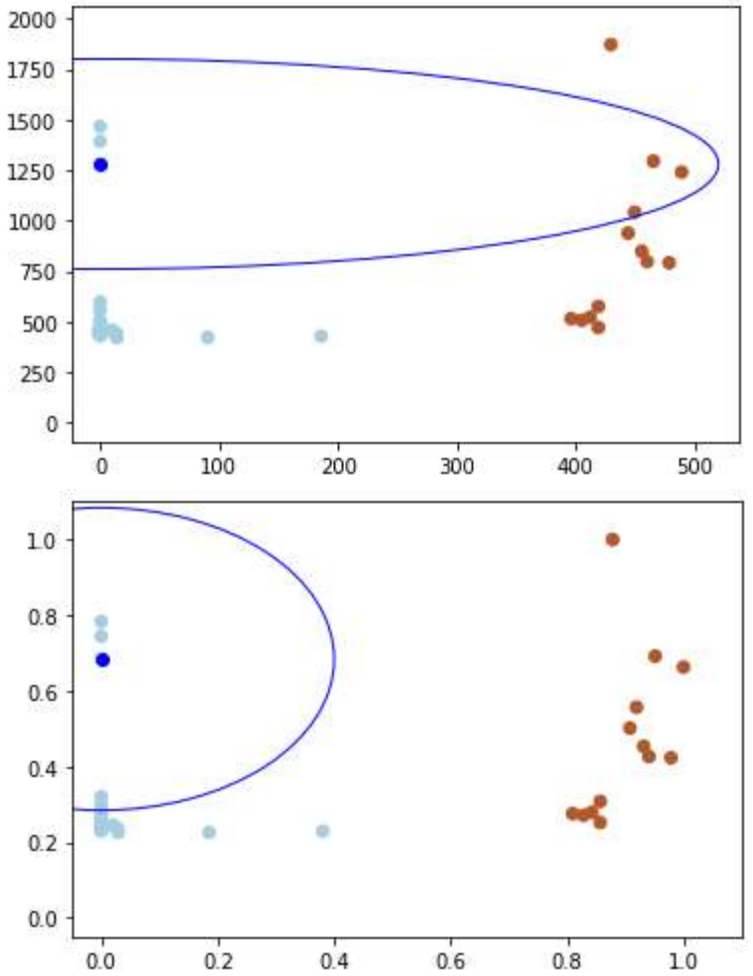
Feature Scaling

For numeric machine learning methods
normalize your input data

https://youtu.be/gV5fD8Xbwgk

Pictured: kNN classification
Ellipse is k- neighborhood of dark blue point

(brown data points from different class included due to scale distortion)



Least Squares

Method 2: with two dimensions, one variable x:

$$w_1 = \frac{N\left(\sum x_j y_j\right) - \left(\sum x_j\right)\left(\sum y_j\right)}{N\left(\sum x^2\right) - \left(\sum x\right)^2}$$

$$w_0 = \frac{1}{N}\left(\sum y_j - w_1 \sum x_j\right)$$

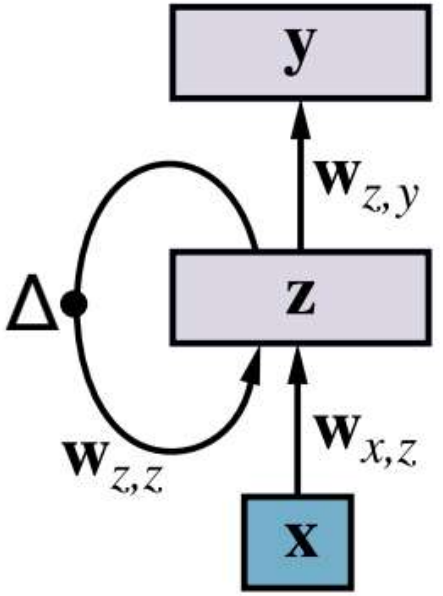
Minimizes the loss L for $\hat{y} = w_0 + w_1 x$

(Include all x-s and y-s from N samples when computing)

Recurrent Neural Nets

Internal State

Basic RNN layout:

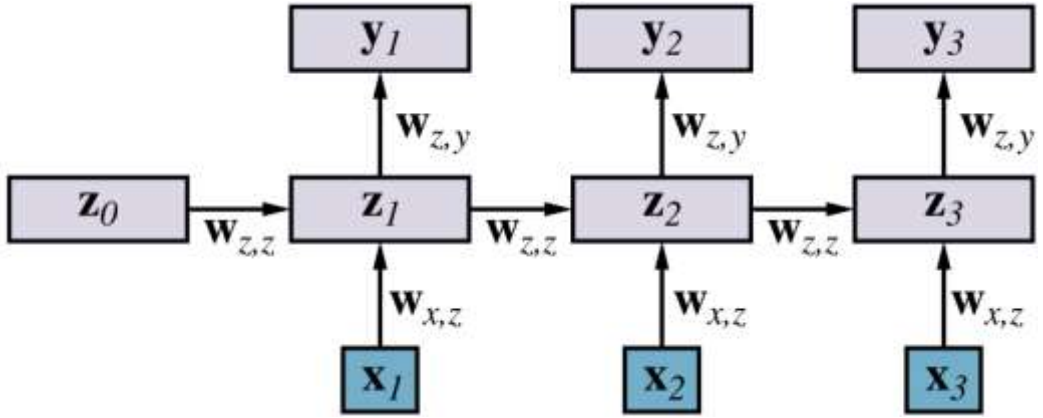


Inputs are a sequence $x_1, x_2, \ldots$
At time step t, calculate hidden layer z_t
from current input x_t and z_{t-1}

The feedback loop allows the RNN
to “remember” the previous inputs

RNN Training

Training is tricky! Basic idea is to “unroll” the network in time



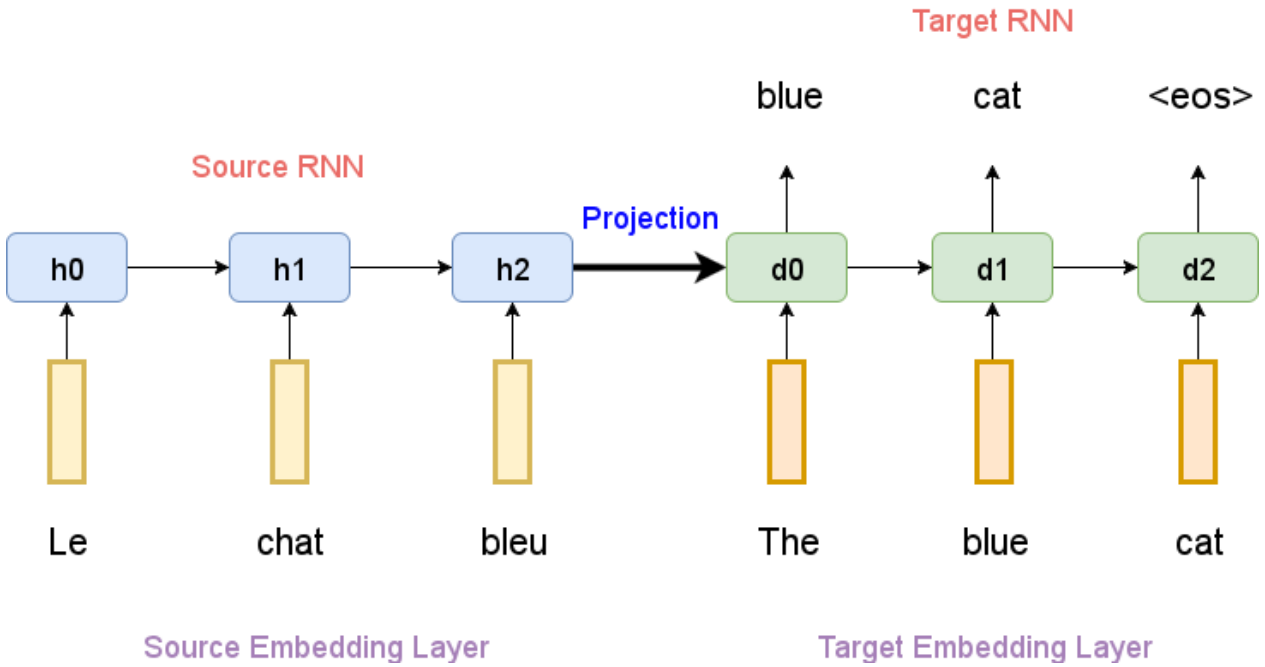
RNN at $t = 3$ represented as a regular feed-forward network

Can be trained with backpropagation (theoretically)

In practice, special architectures – LSTM, GRU for efficient training

Example Application

Machine translation with “seq2seq”



h0, h1, h2
are the **same** encoder RNN (or stack of RNNs) at **different time steps**

d0, d1, d2
decoder RNN
at different time steps

Image: github.com/MaximumEntropy

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." *arXiv preprint arXiv:1409.3215* (2014).

Demo

Basic RNN: input and output are one character

Generate text by feeding its own output as next input

Timestep	1	2	3	4	5	6	7
Input	START	h	e	l	l	o	
Output	h	e	l	l	o		w

https://cs.stanford.edu/people/karpathy/recurrentjs/

Deep Reinforcement Learning

Limits of Q-table

In Q-learning, $Q(s, a)$ can be represented as a table

What if you had millions or billions of different states?

e.g. if input is video feed, we get 4^{7056} states even with a scaled down and monochrome-converted resolution of 84x84.

Mnih, Volodymyr, et al. "Playing Atari with deep reinforcement learning."
arXiv preprint arXiv:1312.5602 (2013).

Linear Approximation

Intermediate idea that has worked for some problems:

$$U(s) = w_1 f_1(s) + \cdots + w_n f_n(s)$$

Utility of state, f_1 to f_n are **features** of the state
Learn the weights $\boldsymbol{w}$ by gradient descent
(Decide actions based on $U(s)$ of expected state)

Similar approximation of $Q(s, a)$ is trickier

Image: IBM Think Magazine, December 1992

Deep Reinforcement Learning

Idea: compute $Q(s, a)$ with a neural network

non-linear, does **automatic** feature selection

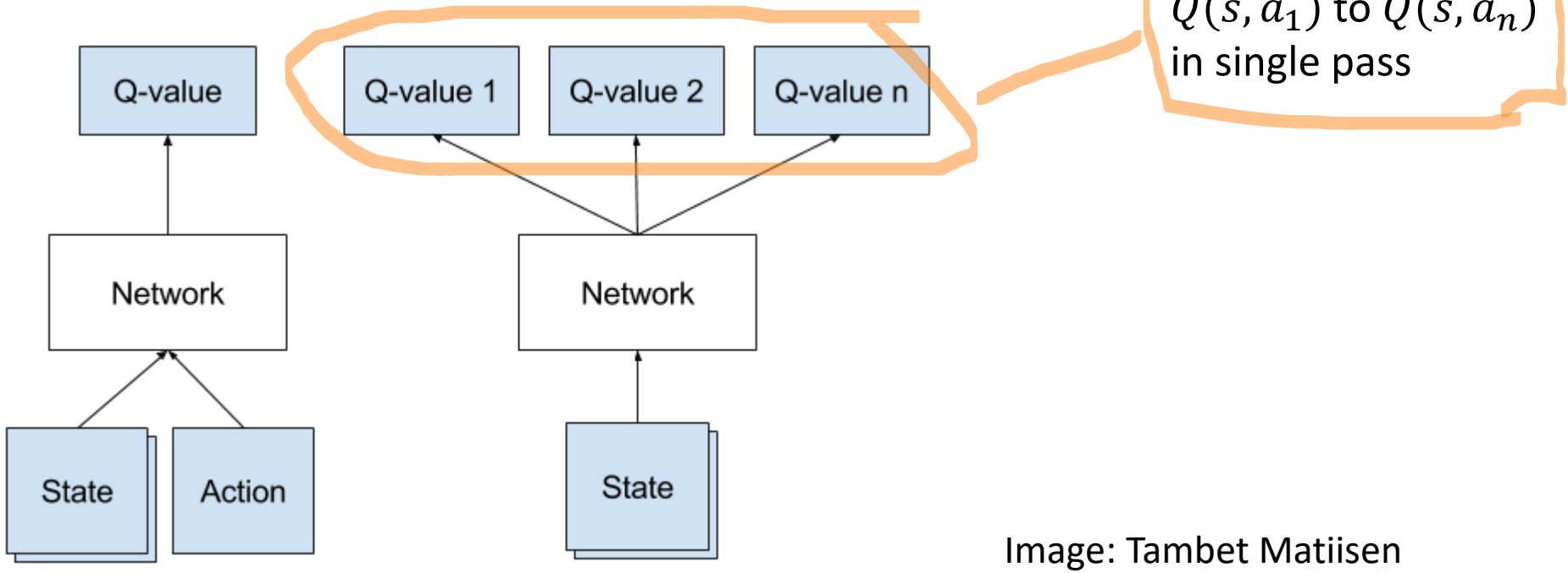


Image: Tambet Matiisen